Practice 05. TRIGGER & OTHER CONSTRAINTS

PART 1. Theory Review

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
BEFORE | AFTER | INSTEAD OF  [INSERT] [,] [UPDATE] [,] [DELETE] [OF
<column_name>] ON <table_name>
[REFERENCING
        OLD ROW|TABLE AS <variable_name>,
        NEW ROW|TABLE AS <variable_name>]
FOR EACH ROW | FOR EACH STATEMENT
[WHEN <condition>]
BEGIN
        <statements>
END;
```

Note: [..]: optional clause
OLD TABLE and NEW TABLE are used in the case FOR EACH STATEMENT (default mode)
OLD ROW and NEW ROW are used in the case FOR EACH ROW
Example:

Given table: `MovieExec(name, address, cert#, netWorth)`

Trigger 1:

```
 1)  CREATE TRIGGER NetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)      OLD ROW AS OldTuple,
 5)      NEW ROW AS NewTuple
 6)  FOR EACH ROW
 7)  WHEN (OldTuple.netWorth > NewTuple.netWorth)
 8)      UPDATE MovieExec
 9)      SET netWorth = OldTuple.netWorth
10)      WHERE cert# = NewTuple.cert#;
```

Trigger 2:

```
 1)  CREATE TRIGGER AvgNetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)      OLD TABLE AS OldStuff,
 5)      NEW TABLE AS NewStuff
 6)  FOR EACH STATEMENT
 7)  WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
 8)  BEGIN
 9)      DELETE FROM MovieExec
10)      WHERE (name, address, cert#, netWorth) IN NewStuff;
11)      INSERT INTO MovieExec
12)          (SELECT * FROM OldStuff);
13)  END;
```

Trigger 3:

```
1)  CREATE TRIGGER FixYearTrigger
2)  BEFORE INSERT ON Movies
3)  REFERENCING
4)      NEW ROW AS NewRow
5)      NEW TABLE AS NewStuff
6)  FOR EACH ROW
7)  WHEN NewRow.year IS NULL
8)  UPDATE NewStuff SET year = 1915;
```

Part II. Practice with MS SQL Server:

SQL basic Trigger structure

```
CREATE [OR ALTER] TRIGGER [schema_name.] trigger_name
ON <table_name>
FOR | AFTER | INSTEAD OF
[INSERT] [,] [UPDATE] [,] [DELETE] [OF <column_name>]
AS <sql statements>
```

Three things to remember:

(1) NO FOR EACH ROW
(2) BEFORE → FOR
(3) NO REFERENCING OLD OR NEW ROW OR TABLE
(4) USE INSERTED and DELETED TABLES for referencing old and new tuples

Operation	*deleted* Table	*inserted* Table
INSERT	(not used)	Contains the rows being inserted
DELETE	Contains the rows being deleted	(not used)
UPDATE	Contains the rows as they were before the UPDATE statement	Contains the rows as they were after the UPDATE statement

Examples

Step 1: create sample tables:

```
CREATE TABLE dbo.SampleTable (
  SampleTableID INT NOT NULL IDENTITY(1,1),
  SampleTableInt INT NOT NULL,
  SampleTableChar CHAR(5) NOT NULL,
  SampleTableVarChar VARCHAR(30) NOT NULL,
  CONSTRAINT PK_SampleTable PRIMARY KEY CLUSTERED (SampleTableID)
);
GO

CREATE TABLE dbo.SampleTable_Audit (
  SampleTableID INT NOT NULL,
  SampleTableInt INT NOT NULL,
  SampleTableChar CHAR(5) NOT NULL,
  SampleTableVarChar VARCHAR(30) NOT NULL,
  Operation CHAR(1) NOT NULL,
```

```
  TriggerTable CHAR(1) NOT NULL,
  AuditDateTime smalldatetime NOT NULL DEFAULT GETDATE(),
);
```

Step 2: Create triggers

```
CREATE TRIGGER dbo.SampleTable_InsertTrigger
ON dbo.SampleTable
FOR INSERT
AS
BEGIN
   INSERT INTO dbo.SampleTable_Audit
   (SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, Operation, TriggerTable)
   SELECT SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, 'I', 'I'
   FROM inserted;
END;
GO

CREATE TRIGGER dbo.SampleTable_DeleteTrigger
ON dbo.SampleTable
FOR DELETE
AS
BEGIN
   INSERT INTO dbo.SampleTable_Audit
   (SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, Operation, TriggerTable)
   SELECT SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, 'D', 'D'
   FROM deleted;
END;
GO

CREATE TRIGGER dbo.SampleTable_UpdateTrigger
ON dbo.SampleTable
FOR UPDATE
AS
BEGIN
   INSERT INTO dbo.SampleTable_Audit
   (SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, Operation, TriggerTable)
   SELECT SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, 'U', 'D'
   FROM deleted;

   INSERT INTO dbo.SampleTable_Audit
   (SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, Operation, TriggerTable)
   SELECT SampleTableID, SampleTableInt, SampleTableChar,
SampleTableVarChar, 'U', 'I'
   FROM inserted;
```

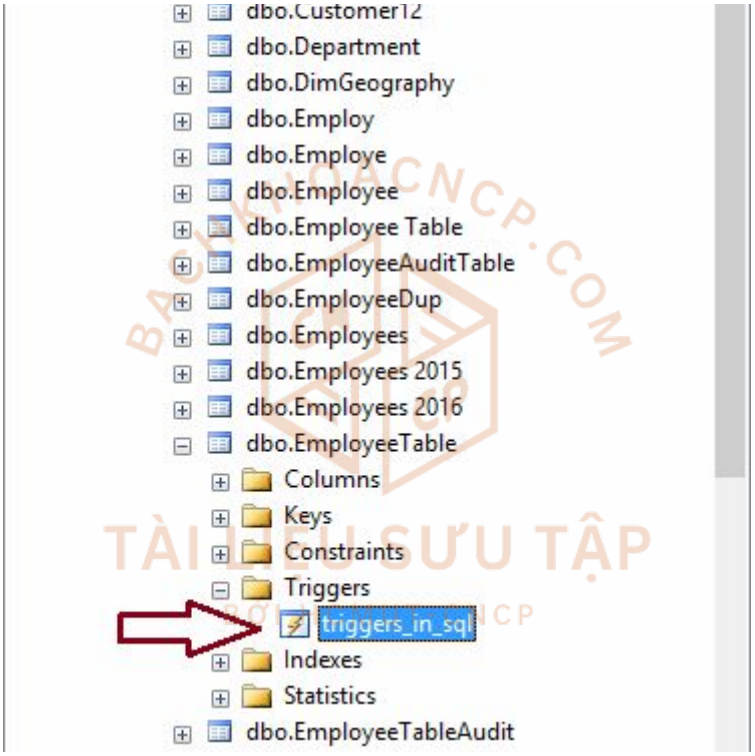
```
END;
```
Now let's test the triggers:

```
-- First the inserts
INSERT INTO dbo.SampleTable
(SampleTableInt, SampleTableChar, SampleTableVarChar)
VALUES
(1, '11111', '1111111111');

INSERT INTO dbo.SampleTable
(SampleTableInt, SampleTableChar, SampleTableVarChar)
VALUES
(2, '22222', '222222222222222');

INSERT INTO dbo.SampleTable
(SampleTableInt, SampleTableChar, SampleTableVarChar)
VALUES
(3, 'AAAAA', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ');
GO

-- Check the sample table
SELECT * FROM dbo.SampleTable;
GO

-- Check the inserts
SELECT * FROM dbo.SampleTable_Audit;
GO

-- Perform a delete operation
DELETE FROM dbo.SampleTable
WHERE SampleTableInt = 2;
GO

-- Check the sample table
SELECT * FROM dbo.SampleTable;
GO

-- Check the delete
SELECT * FROM dbo.SampleTable_Audit;
GO

-- Perform an update operation
UPDATE dbo.SampleTable
SET SampleTableChar = '33333'
WHERE SampleTableInt = 3;
GO

-- Check the sample table
SELECT * FROM dbo.SampleTable;
GO
```

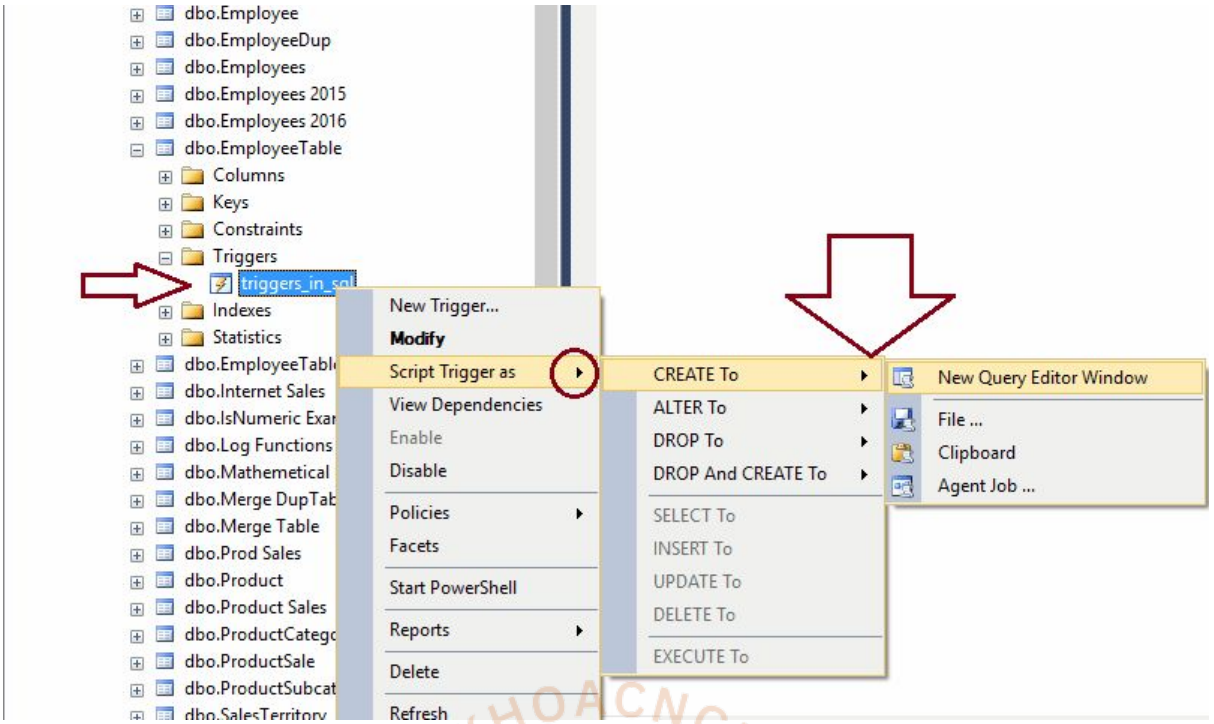
```
-- Check the update
SELECT * FROM dbo.SampleTable_Audit;
GO
```

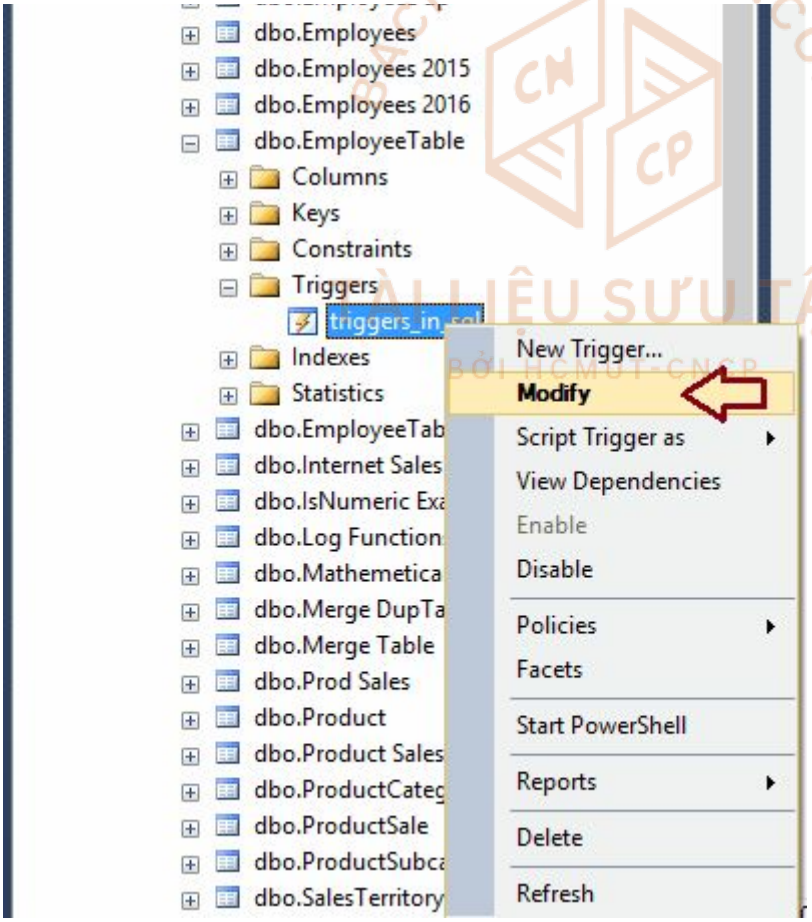
VIEW TRIGGERS CREATED ON TABLES AND SOME ACTIONS ON SSMS

1. Open SQL Server Management Studio and connect to a server
2. Open Object Explorer if not open yet
3. Expand the server node, then databases folder
4. Expand your database
5. Expand tables folder
6. Expand the table you want to check
7. Expand the triggers folder and will see the list of trigger if any.



You can modify, delete, disable or enable the triggers created by SSMS.





Delete SQL Trigger using Query

```
DROP TRIGGER [dbo].[triggers_in_sql]
```

INSTEAD OF TRIGGERS

INSTEAD OF triggers cause their source DML operation to skip and they just execute the code provided inside them. Actual insert, delete or update operation do not occur at all. However they have their associated inserted and deleted tables simulating the DML operation. Inserted and deleted tables are widely used in operations inside triggers and I will show you some examples below . We will discuss further aspects of INSTEAD OF triggers by going through some examples for DML operations.

Example 01:

```
CREATE TABLE BaseTable
  (PrimaryKey     int PRIMARY KEY IDENTITY(1,1),
   Color          nvarchar(10) NOT NULL,
   Material       nvarchar(10) NOT NULL,
   ComputedCol AS (Color + Material)
  )
GO

--Create a view that contains all columns from the base table.
CREATE VIEW InsteadView
AS SELECT PrimaryKey, Color, Material, ComputedCol
FROM BaseTable
GO

--Create an INSTEAD OF INSERT trigger on the view.
CREATE TRIGGER InsteadTrigger on InsteadView
INSTEAD OF INSERT
AS
BEGIN
  --Build an INSERT statement ignoring inserted.PrimaryKey and
  --inserted.ComputedCol.
  INSERT INTO BaseTable
       SELECT Color, Material
       FROM inserted
END
GO
```

Testing your results:

```
--A correct INSERT statement that skips the PrimaryKey and ComputedCol columns.
INSERT INTO BaseTable (Color, Material)
     VALUES (N'Red', N'Cloth')

--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, ComputedCol
```

FROM BaseTable

--An incorrect statement that tries to supply a value for the PrimaryKey and ComputedCol columns.

INSERT INTO BaseTable
    VALUES (2, N'Green', N'Wood', N'GreenWood')

--A correct INSERT statement supplying dummy values for the PrimaryKey and ComputedCol columns.

INSERT INTO InsteadView (PrimaryKey, Color, Material, ComputedCol)
    VALUES (999, N'Blue', N'Plastic', N'XXXXXX')

--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, ComputedCol
FROM InsteadView

OTHER PRACTICE

1. Create table according to the below information

EmployeeTable (ID, Name, Education, Occupation, YearlyIncome, Sales)
EmployeeAuditTable(ID, Name, Education, Occupation, YearlyIncome, Sales, ServerName, ServerInstanceName, InsertTime)

2. Create trigger

```
-- Example for INSTEAD OF INSERT Triggers in SQL Server
CREATE TRIGGER InsteadOfINSERTTriggerExample on [EmployeeTable]
INSTEAD OF INSERT
AS
BEGIN
INSERT INTO [EmployeeTableAudit](
      [ID],
      [Name],
      [Education],
      [Occupation],
      [YearlyIncome],
      [Sales],
         [ServerName],
      [ServerInstanceName],
      [Insert Time])
SELECT  ID,
         Name,
      Education,
      Occupation,
      YearlyIncome,
```

```
      Sales,
      CAST( SERVERPROPERTY('MachineName') AS VARCHAR(50)),
      CAST( SERVERPROPERTY('ServerName') AS VARCHAR(50)),
      GETDATE()
FROM INSERTED
WHERE YearlyIncome <= 70000;
PRINT 'We Successfully Fired Our INSTEAD OF INSERT Triggers in SQL
Server.';

INSERT INTO [EmployeeTable](
      [Name],
      [Education],
      [Occupation],
      [YearlyIncome],
      [Sales])
SELECT  Name,
      Education,
      Occupation,
      YearlyIncome,
      Sales
FROM INSERTED
WHERE YearlyIncome > 70000;
END;
```

3. Lets test your triggers!

====

Exercises:

Use Company database, create triggers as below

1. Employees must have the age greater than 18 when being inserted into database or updated
2. The location of projects must be in the locations of the department which manages it
3. Create the new table employee_totalhours including ssn, total_emp_hours
   Create the trigger which automatically calculates total working hours for employees over their projects.